Uncertainty Averse Pushing with Model Predictive Path Integral Control

Uncertainty Averse Pushing with Model Predictive Path Integral Control

Ermano Arruda*[1], Michael J Mathew*[1], Marek Kopicki[1], Michael Mistry[2], Morteza Azad[1] and Jeremy L Wyatt[1]

I. INTRODUCTION

***Abstract*— Planning robust robot manipulation requires good forward models that enable robust plans to be found. This work shows how to achieve this using a forward model learned from robot data to plan push manipulations. We explore learning methods (Gaussian Process Regression, and an Ensemble of Mixture Density Networks) that give estimates of the uncertainty in their predictions. These learned models are utilised by a model predictive path integral (MPPI) controller to plan how to push the box to a goal location. The planner avoids regions of high predictive uncertainty in the forward model. This includes both inherent uncertainty in dynamics, and meta uncertainty due to limited data. Thus, pushing tasks are completed in a robust fashion with respect to estimated uncertainty in the forward model and without the need of differentiable cost functions. We demonstrate the method on a real robot, and show that learning can outperform physics simulation. Using simulation, we also show the ability to plan uncertainty averse paths.**

II. INTRODUCTION

Manipulating objects via non-prehensile actions, such as pushing, is a well-known problem in robotics [6], [10]. Planning these push-manipulations requires a forward model. There are many ways to express and acquire such a model, from analytic mechanics to machine learning, as well as hybrid techniques. There are several open problems, of which we address two. First, push planning typically does not take account of all the types of uncertainty in the predictions of the forward model. Second, when using purely learned models, push planning has only been demonstrated for single pushes, not for complex push sequences. In this work we present a combined solution to these problems.

Uncertainty in prediction comes from two sources. First, it can arise from small variations in physical properties, such as shape and friction, that are hard to measure, but which significantly alter action effects. Second, in a learned forward model it can arise from a paucity of data. In this paper, we use two different learning methods to explicitly predict a distribution over push outcomes, including both types of uncertainty.

When planning an action sequence the robot can take account of regions of high uncertainty. This is important because actions in uncertain regions of the state space can lead to unrecoverable states. We model these as incurring a cost that rises with the uncertainty predicted by the forward model. But, what push planner can we use? The choice is complicated by the fact that the overall cost function is typically not a differentiable function of the actions. In this case, a path integral formulation [15], [28], [29] works well. We also utilise re-planning each step to account for model inaccuracies. Thus, our planner is a model predictive path integral (MPPI) controller that performs uncertainty averse pushing.

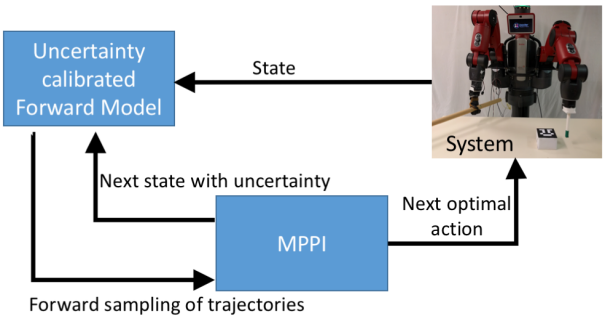


Fig. 1. A high-level diagram of the approach. The forward model is acquired from data, and gives uncertainty in its predictions. The system is the robot plus box and environment.

The technical contributions of this work are: (i) applying ensembles of MDNs and Gaussian Processes to learn uncertainty aware forward models of push manipulation; (ii) using a learnt forward model with model predictive path integral (MPPI) control to push an object to a given goal pose with a many step plan; and (iii) two algorithms for uncertainty averse push planning.

The paper is organised as follows. Section III reviews existing work on push learning and planning. Section IV gives a brief background on the elements of our approach. Section V explains the problem formulation and algorithms. Section VI details the experimental study. Finally, Section VII is a discussion.

III. RELATED WORK

A. Learning Models for Pushing

There are many approaches to modelling push effects based on classical analytic mechanics [22]–[25]. These require modelling and knowledge of parameters such as friction, mass, inertia, and centre of gravity, of the manipulated object, manipulator, and other objects in contact. Accurate identification of these parameters is hard. Even if this is solved, approximations used in rigid body engines can render poor predictions. An alternative is to learn a model from data [1], [11], [18], [26], or to use a hybrid approach [3], [32]. Learning methods divide into data-intensive and data-efficient methods.

Data-intensive methods, such as deep-learning, adopt self-supervision for data collection, allowing the creation of large datasets. In [1], for example, a *siamese* architecture learns a forward and an inverse model for pushing. The forward model is used as a regulariser for the inverse model. Limitations are the use of a discrete action space and the lack of a representation of predictive uncertainty. Finn et al [11] used an auto-encoder based forward model that is used in a model predictive control schema to find push actions based on image input. However, this model also lacks knowledge of the predictive uncertainty and has only been shown to achieve single step push manipulations. Pinto et al. [26], used

*Joint first authors.

We gratefully acknowledge support of the Commonwealth scholarship by the British Council for Michael Mathew and a scholarship from the Brazilian National Council for Scientific and Technological Development (CNPq) for Ermano Arruda.

[1]School of Computer Science, University of Birmingham, B15 2TT, UK (exa371,mjm522,msk,m.azad,jlw)@cs.bham.ac.uk

[2]Edinburgh Centre for Robotics, University of Edinburgh, Edinburgh, EH8 9YL, UK mmistry@inf.ed.ac.uk

push models to improve grasp performance through pushing. Again, predictive uncertainty in the learnt models was not explored.

There are also data-efficient approaches to learning push effects [2], [18], [19]. These models typically use hand-crafted features, and have not yet been used for push planning. They do, however, represent uncertainty in the outcome, including, in [2], the ability to predict meta-uncertainty due to a lack of data.

B. Estimating Predictive Uncertainty

The ability to predict both uncertainty in dynamics and meta-uncertainty in this dynamics model is useful for robot planning [8]. Policy search method PILCO [8] utilises Gaussian Processes (GPs) [29]. GPs, however, scale poorly with the amount of training data. Representations such as Gaussian Mixture Regression scale better and can estimate dynamic uncertainty, but not the meta uncertainty [7]. A neural network approach to representing uncertainty in dynamics is to learn the parameters of a mixture density. This is termed an mixture density network (MDN). This can be extended to add meta-uncertainty due to a lack of data in various ways. One way is to use dropout [12]. Another way is to use an ensemble of MDNs and adversarial training [20]. We use this latter approach.

C. Path Integral Applications for Control

Stochastic optimal control (SOC) deals with both uncertainty in the action and sensor models, and the resultant state uncertainty. The sequence of control commands is found by minimising an integral of individual step costs (called the running cost) along a given trajectory. The SOC problem is defined by a *Hamilton-Bellman-Jacobi* (HJB) partial differential equation (PDE) corresponding to the system to be controlled. This can be solved numerically backwards in time, given the system's initial and target configurations [15]. This is straightforward for linear systems with quadratic costs [21], but non-trivial for non-linear systems.

However, by using the Feynman-Kac theorem, a non-linear HJB can be converted into a linear PDE, which can be solved via forward sampling of trajectories [15], [28]. This formulation can cope with arbitrary state costs that need not be differentiable, and is applicable to a wide range of non-linear systems. Recently, researchers have explored its benefits for robot control [29]–[31]. We make use of the path integral framework. Specifically, we apply the model predictive path integral control algorithm proposed by [29].

IV. PRELIMINARIES

Classical optimal control deals with finding a set of control actions that solves the problem (typically deterministic systems) and is optimal with respect to a cost function. The general framework is to design an agent as an automaton that seeks to minimise a cost function for a fixed or varying time horizon [16]. There are typically two methods to solve an optimal control problem. They are the HJB formulation (finding a solution using dynamic programming) and the second is by using *Pontryagin's Minimum Principle* (PMP) (finding a solution to the ordinary differential equations formed) [17]. These formulations are typically interested in finding a globally optimal solution.

In this paper we define a trajectory as being a sequence of states $\mathbf{x}_t$, actions $\mathbf{u}_t$ and associated uncertainty $\hat{\sigma}_{t,n}$ at time step t. Thus, let $\mathbf{s}_{t,n}$ be a convenient tuple, such that $\mathbf{s}_{t,n} = (\mathbf{x}_{t,n}, \mathbf{u}_{t,n}, \hat{\sigma}_{t,n})$. Then, using n to index a trajectory and i to index a discrete timestep, the optimal control problem is cast by defining a cost function for a trajectory n starting from timestep i until T, i.e. $\tau_{i,n} = [\mathbf{s}_{i,n}, \mathbf{s}_{i+1,n}, \mathbf{s}_{i+2,n}, ..., \mathbf{s}_{T,n}]$:

$$S_i = S(\tau_{i,n}) = \phi_T(\mathbf{x}_T) + \sum_{t=i}^{T-1} r_t(\mathbf{x}_t, \mathbf{u}_t, \hat{\sigma}_t) \tag{1}$$

where, r_t is the immediate cost function and ϕ_T is the final cost function. The aim is to find a policy that minimises the above cost function. The value function for a state is defined as the minimum cumulative cost the agent can obtain from a state, if it proceeds optimally from that state to the goal. The value function for a state $\mathbf{x}_i$ can be defined as:

$$V(\mathbf{x}_i) = \min_{\mathbf{u}_{i:T}} E[S_i] \tag{2}$$

At any state $\mathbf{x}_i$, the aim is to find a set of control commands or actions $\mathbf{u} = (\mathbf{u}_i, ...\mathbf{u}_T)$ that would minimise the expected cumulative cost from that state. Note that in Eq 2 the expectation is taken over all possible paths (i.e. trajectories) and thus the index n is dropped for convenience.

V. APPROACH

We now describe the proposed uncertainty averse push planner, which comprises two parts. First, an uncertainty calibrated forward model is learnt with data collected from a variety of pushes. Second, using the learnt model, we formalise our planning problem as Model Predictive Path Integral control (MPPI) and detail our planning algorithm. Later we describe another path integral based approach to find a low cost trajectory that can be followed by using the MPPI controller.

A. Forward Models with Predictive Uncertainty

There are various ways to capture uncertainty in predictions. Gaussian Processes, for example, provide an effective and theoretically clean tool to make uncertainty aware predictions [27]. The main problem with Gaussian processes (GPs) is their inability to scale to high dimensional spaces or large data sets. We therefore also utilise an ensemble of mixture density networks (E-MDN).

Arbitrary densities can also be learnt via gradient descent with Mixture Density Networks (MDNs) [4]. In such models, if we choose the mixture components to be Gaussian, the network outputs the parameters of a Gaussian mixture model conditioned on a suitable choice of input vector $\mathbf{h} \in \mathcal{R}^n$, thus modelling arbitrary multi-modal densities as defined in equation 3.

$$p_\theta(\mathbf{x}_{t+1}|\mathbf{h};\theta) = \sum_{k=1}^{K} \pi_k(\mathbf{h};\theta)\mathcal{N}(\mathbf{x}_{t+1}|\mu_\mathbf{k}(\mathbf{h};\theta), \sigma_k^2(\mathbf{h};\theta)), \tag{3}$$

where $\pi_k(\mathbf{h};\theta)$, $\mu_k(\mathbf{h};\theta)$ and $\sigma_k(\mathbf{h};\theta)$ are the network outputs which form the parameters of the mixture. In order to make sure the network outputs valid parameters for the mixture, Bishop [4] suggests using a softmax layer to represent π_k, such that $\sum_{k=1}^{K} \pi_k(\mathbf{h};\theta) = 1$, whereas an exponential layer is able to guarantee that σ_k is positive definite, and finally μ_k can be a linear combination of hyperbolic tangent activation functions. The reader is encouraged to refer to Bishop [4] and Graves [13] for further details on practical considerations in implementing MDNs.

Concretely, for learning a forward model, we define $\mathbf{h} = [\mathbf{x}_t, \mathbf{u}_t]$, where $\mathbf{x}_t$ is the state at time step t, (position and orientation of the box on the plane) $\mathbf{x}_t = [x_t, y_t, \theta_t]$, and $\mathbf{u}_t$ is the action taken at that time, encoded as a direction vector $[px_t, py_t]$ and a single real value a_t, normalised between zero and one, indicating the contact location on the box edge, i.e. $\mathbf{u}_t = [px_t, py_t, a_t]$ (all quantities are given in the object frame). Furthermore, as has been demonstrated

by [20], one can form an ensemble of such models so as to estimate the uncertainty of the forward model's predictive distribution. If an ensemble is composed of M members, the final model can be written as:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t; \theta_m) \tag{4}$$

The statistics of interest that we compute from the ensemble are the mean prediction and the variance, which, when trained accordingly, can reflect the predictive uncertainty of the model [20], and are given by:

$$\hat{\mu} = \frac{1}{M}\sum_{m=1}^{M} \mu_{\theta_m} \tag{5}$$

$$\hat{\sigma}^2 = \frac{1}{M}\sum_{m=1}^{M} (\sigma^2_{\theta_m} + \mu^2_{\theta_m}) - \hat{\mu}^2 \tag{6}$$

Thus, the predictive uncertainty we refer throughout this paper represents both inherent uncertainty in dynamics, and meta uncertainty due to lack of data, as given by 6.

B. Uncertainty Averse Model Predictive Path Integral Control

Once we have a forward model, we need a way to use this model to find the right sequence of push commands to move the object to the goal. When solving a task, it is easier to exploit the already known part of the state-action space rather than exploring new parts. This suggests moving through more certain regions of the state-action space. We use the path integral based approach to model predictive control in [29].

The path integral formulation for stochastic optimal control permits one to find policy updates by calculating expectations over trajectory roll outs [28]. It provides an alternative to directly solving the non-linear HJB equation via backward integration, and allows one to find the command updates that minimise the cost-to-go in Eq 7 as a weighted average over N forward sampled trajectories.

$$S(\tau_{i,n}) = \phi(\mathbf{x}_T) + \sum_{t=i}^{T-1} r(\mathbf{x}_t, \mathbf{u}_t, \hat{\sigma}_t), \tag{7}$$

Given the cost-to-go in Eq 7, one wants to find the optimal action sequence as $\overset{*}{\mathbf{u}} = \arg\min_{\mathbf{u}} E[S_i]$. Note that in this work the cost is also a function of the predictive uncertainty $\hat{\sigma}_t$, in addition to the state $\mathbf{x}_t \in \mathcal{R}^n$ and controls $\mathbf{u}_t \in \mathcal{R}^m$. The importance of each n^{th} sample is given by a weight, defined as the exponential of the cost-to-go $S(\tau_{i,n})$, which is given by:

$$w_{i,n} = \frac{\exp(-\frac{1}{\lambda}S(\tau_{i,n}))}{\sum_{l=1}^{N} exp(-\frac{1}{\lambda}S(\tau_{i,l}))} \tag{8}$$

where λ can be seen as the temperature parameter for the softmax distribution in Eq 8 and affects the control update given by Eq 9.

$$\Delta\mathbf{u}_i = \sum_{n=0}^{N} w_{i,n}\delta\mathbf{u}_{i,n}(\eta) \tag{9}$$

Thus, the control command updates are calculated as an expectation, or weighted average, over sampled control disturbances $\delta\mathbf{u}_{i,n}$ with weights equal to $w_{i,n}$. Here, control disturbances are in a similar manner to that defined in [29]. However, we introduce an exploration decay parameter η, i.e.

$$\delta\mathbf{u}_{i,n}(\eta) = \eta\frac{1}{\sqrt{\rho}}\frac{\epsilon}{\sqrt{\Delta t}}, \tag{10}$$

where Δt is the time step magnitude, with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which has same dimensionality as the control actions $\mathbf{u_t} \in \mathcal{R}^m$. The parameter ρ can be seen as a constant responsible for controlling the magnitude or level of exploration of the sampled disturbances $\delta\mathbf{u}_{i,n}$, whereas a suitable exploration decay schedule for η helps to ensure local convergence even when the number N of sampled trajectories is small. In all experiments presented in this paper η is always initialised as $\eta = 1.0$ and geometrically decays over a chosen number of decay steps L as detailed in Algorithm 1.

By using a model predictive approach we are able to incorporate feedback into the system. At each state a look-ahead window of T time steps is used, starting at time step i. Then the first control command of the T steps is executed on the robot. After this first push, the new state is fed back into the optimiser and the process is repeated till task convergence.

The immediate cost function used in our formulation is

$$r(\mathbf{x}_i, \mathbf{u}_i, \hat{\sigma}_i) = \gamma * \hat{\sigma}_i + (\mathbf{x}_i - \mathbf{x}_{goal})^T Q(\mathbf{x}_i - \mathbf{x}_{goal}) + \mathbf{u}_i^T R\mathbf{u}_i, \tag{11}$$

and the final cost is given by

$$\phi(\mathbf{x}_T) = (\mathbf{x}_T - \mathbf{x}_{goal})^T Q(\mathbf{x}_T - \mathbf{x}_{goal}) \tag{12}$$

By adding the $\hat{\sigma}$ term in equation 11, the samples that pass through an uncertain region are penalized more and hence would contribute less to the control update in equation 9. Throughout the remainder of the paper, the state of the object to be pushed is defined by its position and orientation on the plane under the quasi-static assumption, subject to only planar motion, i.e. $\mathbf{x}_i = [x_i, y_i, \theta_i]$.

C. An Alternative Approach to Uncertainty Averse Planning

The performance of MPPI for uncertainty averse planning depends on the cost function defined. The challenge of the cost function defined in equation 7 is to optimise the trade off between $\mathbf{Q}$ and γ.

There is a different approach that involves decoupling goal finding and uncertainty reduction. First, a simple path is defined from start to goal. Then the learnt forward model can be used to find a low uncertainty variation of this path. Finally, this can be given to the model predictive controller to follow. The decoupling of the uncertainty cost and the final goal cost gives a significant improvement in performance. We propose a path integral based approach to find a low cost trajectory in the state-action space that can then be followed by the MPPI. The formulation derives inspiration from the STOMP planner [14], though our formulation uses a different cost function and improvement equations. Apart from finding a low cost trajectory, kinematic constraints are imposed on the optimisation problem to find paths within the workspace of the system.

The problem of finding a low uncertainty trajectory can be formulated as:

$$\begin{aligned} &\text{Minimise: } S(\tau_{i,n}) = \alpha\sum_{i=0}^{T}\hat{\sigma}_i \\ &\text{Subject to state constraints: } \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max} \end{aligned} \tag{13}$$

where, for a state $\mathbf{x}$ at time i the uncertainty predicted is σ_i and α is a positive constant. The uncertainty for each point in the heat-map is estimated via Monte Carlo sampling. Thus, by iterating over a subset of states (e.g. states on a grid) and random actions for each state, the average uncertainty is estimated using the learnt forward model with Eq 6 for each state. With this uncertainty heat-map,

Algorithm 1 The modified MPPI algorithm with exploration decay, a modification of the the original algorithm proposed by [29] that uses an uncertainty calibrated forward model (Eq 5 and 6)

Given:
N: Number of samples;
T: Number of timesteps;
L: Number of decay steps;
$(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_T)$: Initial action sequence;
η_{init}: Initial η, for exploration decay, Eq 10;
$\Delta t, \mathbf{x}_{init}, \hat{\mu}(\cdot,\cdot), \hat{\sigma}(\cdot,\cdot)$: System sampling dynamics;
$\phi, r, \mathbf{R}, \mathbf{Q}, \lambda, \gamma$: Cost parameters;
$\mathbf{u}_{init}$: Value to initialise new controls to;

while *task not completed* **do**
  $\eta = \eta_{init}$
  for $l = 0$ **to** L **do**
    for $n = 1$ **to** N **do**
      $\mathbf{x_0} = \mathbf{x}_{init}$
      $\hat{\sigma_0} = \hat{\sigma}(\mathbf{x}_0, \mathbf{0})$ (given by Eq 6)
      for $i = 1$ **to** $T - 1$ **do**
        $\mathbf{x}_{i+1} = \hat{\mu}(\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_{i,n}(\eta))$ (given by Eq 5)
        $\hat{\sigma_{i+1}} = \hat{\sigma}(\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_{i,n}(\eta))$ (given by Eq 6)
        $S(\tau_{i+1,n}) = S(\tau_{i,n}) + r(\mathbf{x_i}, \mathbf{u_i}, \hat{\sigma_i})$
      end for
      $S(\tau_{T,n}) = \phi(\mathbf{x_T})$
    end for
    for $i = 1$ **to** T **do**
      $\mathbf{u}_i = \mathbf{u}_i + \Delta\mathbf{u}_i$ (with $\Delta\mathbf{u}_i$ given by Eq 9)
    end for
    $\eta = 0.99\eta$ (exploration decay)
  end for
  send_control_command($\mathbf{u}_0$)
  for $i = 1$ **to** $T - 1$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$
  end for
  $\mathbf{u}_{T-1} = \mathbf{u}_{init}$
  Update current state
  Check task completion
end while

Algorithm 2 starts with a initial trajectory leading from start to goal. This could be simple linear interpolation from start to goal. If the total number of states in this interpolation is T, then the optimisation is performed for states from 1 to $T-1$, thus ensuring the path continues to reach the goal from the start. The initial trajectory is improved iteratively by usinq Eq 8 for the cost defined in Eq 13. The iterative update is defined by:

$$\Delta\mathbf{x}_i = \sum_{n=0}^{N} w_{i,n}\delta\mathbf{x}_{i,n} \tag{14}$$

VI. EXPERIMENTS

Experiments 1 and 2 described below validate the fundamental ability to push objects to a goal location using E-MDN as the learnt forward model on a real robot (Baxter), without considering uncertainty in the cost function. Experiments 3 and 4 demonstrate in simulation that the same approach is able to find uncertainty averse trajectories for pushing an object to a given goal. For the experiments in the real robot, we found that uncertainty averse pushes achieved in simulation were sometimes hard to reproduce

Algorithm 2 Algorithm 2 optimises a trajectory with respect to model predictive uncertainty. The output of the algorithm is a trajectory with low uncertainty cost which can be tracked by a push controller, in this work we use the Model Predictive Path Integral Controller for following the pushing trajectory.

Given:
N: Number of samples;
T: Number of time steps;
$f(\mathbf{x}, \mathbf{u})$: Learnt forward dynamics;
ϵ: Threshold of cost change;
τ: $(\mathbf{x}_0, \mathbf{x}_2, ..., \mathbf{x}_{T-1})$;
α: positive constant multiplier to cost;
while $S \geq \epsilon$ **do**
  for $n = 0$ **to** N **do**
    for $i = 1$ **to** $T - 1$ **do**
      $\delta\mathbf{x}_{i,n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
      $\mathbf{x}_{i,n} = \mathbf{x}_i + \delta\mathbf{x}_{i,n}$
      Enforce_state_constraints($\mathbf{x}_{n,i}$)
      $S(\tau_{i+1,n}) = S(\tau_{i,n}) + \frac{\alpha}{T}\hat{\sigma_i}$
    end for
  end for
  for $i = 1$ **to** $T - 1$ **do**
    $\mathbf{x}_i = \mathbf{x}_i + \Delta\mathbf{x}_i$ (with $\Delta\mathbf{x}_i$ given by Eq 14)
    Enforce_state_constraints(x_i)
  end for
end while

on the Baxter platform due to kinematic limitations of the robot, i.e. the inverse kinematic solver sometimes failed to find solutions for planned pushes. Thus, general pushing is shown on the real robot, whereas uncertainty averse pushing (which requires using a much larger workspace) is demonstrated in simulation only. We now proceed to describe the experiments in the real robot. Finally, we will describe the simulation experiments that demonstrate uncertainty averse pushing.[1]

A. Experiment 1:

In this experiment we trained the E-MDN model on a set of 326 pushes, gathered from the Baxter robot [2]. The E-MDN utilised had $M = 10$ members in the ensemble, each member being an MDN with 3 hidden layers with 20 neurons per layer. The number of mixtures in each MDN was chosen to be $K = 1$ for simplicity. The model was trained for 3000 epochs with stochastic gradient descent using the Adam [9] optimiser. We utilised batch size 5 and the learning rate was 0.001. Following the training protocol described by [20], we used 0.005 as the adversarial coefficient for generating adversarial examples for training. Figure 2(a) and (b) shows the predictions given by the trained E-MDN model and the GP model respectively.

To show that uncertainty rises when the amount of available data is limited we also gathered data from randomised pushes in Box-2D [5]. Then we lesioned the data for various parts of the state space, and trained the E-MDN model. The uncertainty should be higher in the lesioned parts of space, and this is what we see in Figure 4, in which the average uncertainty for a given state was obtained via

[1]See video summarising the proposed approach and experiments: `https://youtu.be/LjYruxwxkPM`

[2]see data collection setup, in which the robot applies random pushes to the object and periodically restarts the box to an initial location: `https://youtu.be/pRDvkDkCSTQ`

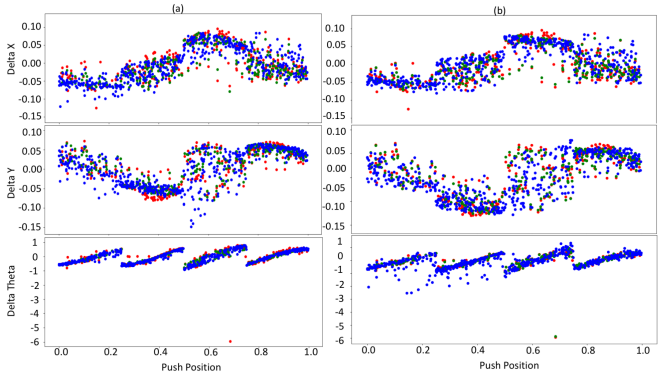


Fig. 2. (a) Predictions of the E-MDN. (b) Predictions of the GP. The top row shows predicted X displacement. The middle shows Y displacement. Theta shows the change in orientation. The red dots illustrate the true delta, the green dots show the prediction on the training set. The blue dots show the prediction on a test set generated from a distribution having same mean and covariance as the training set.

Treatment	Starting pose	Initial cost	Final cost	Steps
MPPI-Box2D	Pose 1	0.800	0.112	13
MPPI-E-MDN	Pose 1	0.795	0.057	8
MPPI-GP	Pose 1	0.764	0.255	12
MPPI-Box2D	Pose 2	0.766	0.097	12
MPPI-E-MDN	Pose 2	0.768	0.079	8
MPPI-GP	Pose 2	0.729	0.072	9

TABLE I

THE COST IS A WEIGHTED AVERAGE OF THE POSITION ERROR (METRES) AND ORIENTATION ERROR (RADIANS).

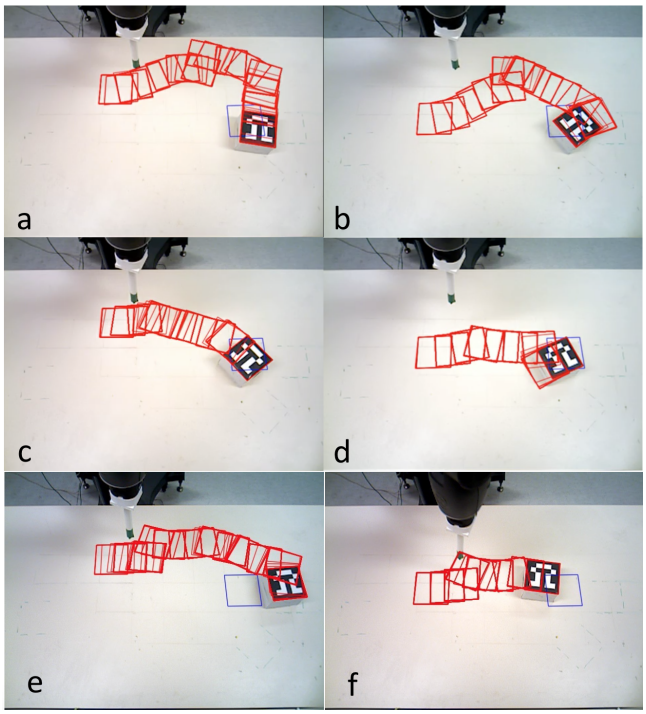


Fig. 3. Push sequences from two positions with MPPI. The first row contains results for Box2D (a and b), the second row contains the results for E-MDN (c and d) and the third row contains results for GP (e and f). Red frames show the starting and intermediate positions, the blue frame shows the goal. In figures a and c the goal has the same orientation as but different position from the initial box pose, whereas in b and d the goal is orientated 90 degrees anti-clockwise with respect to the initial box pose. In our experiments MPPI-E-MDN reached the goal with fewer pushes.

marginalisation over the action space at the given state (average uncertainty for a given box state $\mathbf{x}$ was calculated using Monte-Carlo action samples).

B. Experiment 2:

We performed experiments with the real robot, using the MPPI planner to plan with the learnt models from Experiment 1, and also with a physics simulator suitable for planar pushing called Box-2D [5]. We use this to investigate whether the push planning framework can be combined with a variety of forward models to achieve many-step push manipulations. Some push sequences are visualised in Figure 3. These show that both learning methods terminate with positions close to, but not at, the goal, in terms of both orientation and position. Table I shows that both substantially reduce the cost in paired trials, and that there is no clear difference in performance between the different predictors underpinning MPPI. The cost parameters for the MPPI were chosen to be $\mathbf{Q} = diag(1.5, 1.5, 0.01)$, $\gamma = 0$, $\mathbf{R} = \mathbf{0}$. The optimisation horizon was set to $T = 2$ and the number of sampled trajectory rollouts was chosen to be $N = 150$, $L = 20$, with $h = 1$, $\rho = 1.0$, and $\Delta t = 0.05$.

C. Experiment 3:

Utilising Algorithm 1, together with the cost function defined by Equation 11 set to penalise uncertainty for 150 pushes, and having the uncertainty penalty switched off there after. The aim of this experiment was to show in simulation that, with the right trade-off between the goal and the uncertainty gains, a box can be pushed to a desired location while avoiding regions with high uncertainty. The E-MDN was trained with 261 pushes collected from simulation. The parameters of the model were $M = 10$, in which each MDN member of the ensemble had a single hidden layer with 25 units, and the number of mixtures was set to $K = 1$. The MPPI parameters were set to $\mathbf{Q} = diag(0.5, 0.5, 0.5)$, $h = 2.0$, $\rho = 2.0$, $T = 2$, $\Delta t = 0.05$, $N = 10$, $L = 0$ (exploration decay not utilised) and the uncertainty penalty was set to $\gamma = 115$ for 150 pushes, and then $\gamma = 0$ afterwards. The results for this first experiment are shown by Figures 5 (a) and (b), in which two distinct trajectories are obtained as a result of either penalising or not penalising for model uncertainty.

D. Experiment 4:

Finally, this experiment makes use of Algorithm 2, which performs optimisation to find a low uncertainty cost trajectory first. This low uncertainty cost trajectory is then followed by Algorithm 1 push controller, but this time, we do not need to penalise uncertainty in its running cost, since the trajectory has already been optimised for that. The results for this experiment are shown in Fig 6.

VII. DISCUSSION

A push planning approach that uses a learnt forward model is presented. The push planner is also capable of taking into account the reliability of the learnt model. Initially we showed how a learnt forward model can be used by a real robot to push the object to a target location using the MPPI approach. Later, we showed the modification to this basic MPPI to accommodate predictive uncertainty in two ways. In the first algorithm the uncertainty is directly inserted into the MPPI cost function. In the second formulation, a trajectory that is uncertainty averse is pre-computed using a path-integral update (Algorithm 2) and

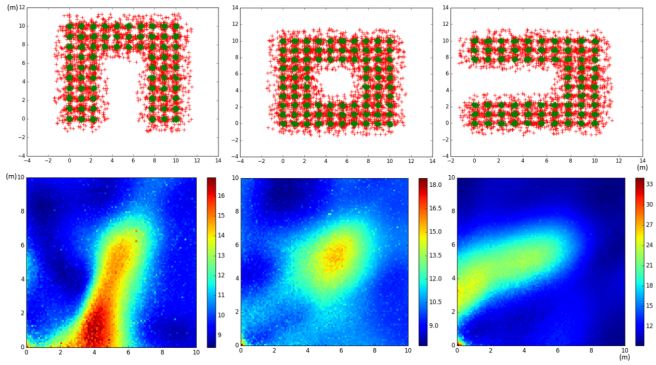
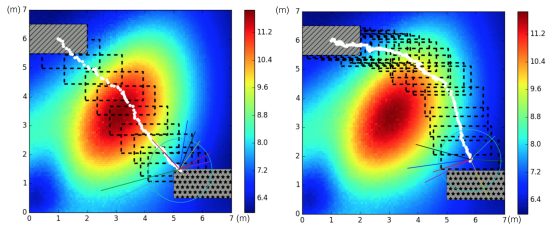
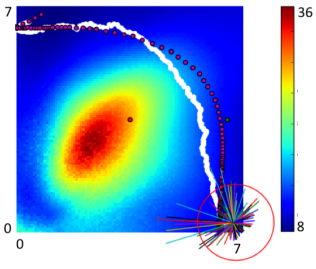


Fig. 4. Simulated data collected from random pushes in Box2D and then lesioned in different ways (left). On the right are the corresponding heat maps depicting forward model model predictive uncertainty in different regions of the state space (right). Starting locations are shown as green crosses, positions after pushes are shown as red crosses.



(a) (b)

Fig. 5. Uncertainty averse pushing (Algorithm 1). The top and bottom grey rectangles depict the start and goal locations, respectively. Dashed boxes in black show sub-sampled box locations along the trajectory in white. In simulation the box is pushed at its CoR, so it will not rotate. (a) Without an uncertainty penalty, the box follows a straight line towards the target. (b) Penalizing uncertainty, the box avoids it. The heuristic cost is defined such that uncertainty acts as a penalty for 150 pushes, and is set to zero afterwards ($\gamma = 0$). For this experiment $\gamma = 115$, $\mathbf{Q} = diag(0.5, 0.5, 0.5)$, $h = 2.0$, $\rho = 2.0$, $T = 2$, $\Delta t = 0.05$, $N = 10, L = 0$. As before, the multi-coloured lines are forward sampled trajectories from the current box state, and the circle radius represents current uncertainty in dynamics.



6. Uncertainty averse pushing (Algorithm 2). The red dots are the way-points to be followed and are generated by Algorithm 2. Algorithm 1 then attempts to follow these, producing the actual trajectory (white line). Forward sampled trajectories from the current box state are shown as multi-coloured lines, and the circle radius represents current region uncertainty.

MPPI (Algorithm 1) is used to follow it. We have shown that both algorithms exhibit the desired behaviour subject to tuning. In addition we have created the data gathering framework on a Baxter robot, and shown that E-MDNs and GPs produce very similar estimates of model uncertainty for real data. Experiments showed that Algorithm 1 works on the real robot, and that either one or both learning methods outpeformed a physics simulator, when used as the forward model for planning.

References

[1] P Agrawal, A Nair, P Abbeel, J Malik, and S Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR-2016*.

[2] M Bauza and A Rodriguez. A probabilistic data-driven model for planar pushing. In *Proc. of IEEE, ICRA-2017*.

[3] D Belter, M Kopicki, S Zurek, and J Wyatt. Kinematically optimised predictions of object motion. In *Proc. of the IEEE, IROS-2014*.

[4] C M Bishop. Mixture density networks. Technical report, 1994.

[5] Erin Catto. Box2D: A 2D physics engine for games, 2011.

[6] A Cosgun, T Hermans, V Emeli, and M Stilman. Push planning for object placement on cluttered table surfaces. In *Proc. of the IEEE, IROS- 2011*.

[7] B Da Silva, G Konidaris, and A Barto. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012.

[8] M Deisenroth and C E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML-2011*.

[9] D Kingma and J Ba. Adam: A method for stochastic optimization. *CoRR-2014*.

[10] M Dogar and S Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and Systems VII*, 2011.

[11] C Finn, I J. Goodfellow, and S Levine. Unsupervised learning for physical interaction through video prediction. *CoRR-2016*.

[12] Y Gal and Z Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML-2016*.

[13] A Graves. Generating sequences with recurrent neural networks. *CoRR-2013*.

[14] M Kalakrishnan, S Chitta, E Theodorou, P Pastor, and S Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Proc. of the IEEE, ICRA-2011*.

[15] H J Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: theory and experiment*, 2005.

[16] H J Kappen. Optimal control theory and the linear Bellman equation. 2011.

[17] D E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

[18] M Kopicki, S Zurek, R Stolkin, T Moerwald, and J L. Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots-2016*.

[19] M Kopicki, S Zurek, R Stolkin, T Mörwald, and J L Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Proc. of the IEEE, ICRA-2011*.

[20] B Lakshminarayanan, A Pritzel, and C Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[21] W Li and E Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO-2004*.

[22] K Lynch. The mechanics of fine manipulation by pushing. In *Proc. of the IEEE, ICRA-1992*.

[23] M T Mason. *Manipulator grasping and pushing operations*. PhD thesis, MIT, 1982.

[24] M T Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

[25] M A Peshkin and A C Sanderson. The motion of a pushed, sliding workpiece.

[26] L Pinto and A Gupta. Learning to push by grasping: Using multiple tasks for effective learning. *CoRR-2016*.

[27] C Rasmussen, E. Gaussian processes for machine learning. 2006.

[28] E Theodorou, J Buchli, and S Schaal. A generalized path integral control approach to reinforcement learning. *JMLR-2010*.

[29] G Williams, A Aldrich, and E Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

[30] G Williams, P Drews, B Goldfain, J M Rehg, and E A Theodorou. Aggressive driving with model predictive path integral control. In *IEEE, ICRA-2016*.

[31] C Yevgen, M Kalakrishnan, A Yahya, A Li, S Schaal, and S Levine. Path integral guided policy search. *CoRR-2016*.

[32] J Zhou, J A Bagnell, and M T Mason. A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation. In *RSS-2017*.